Program Reasoning

8. Automated Program Verification

Kihong Heo

KAIST

Towards Fully Automated Verification

- The assumption so far: a user provides inductive invariants

- Fully automated verification: combined with automated invariant generation methods

- Example:

  - Program analysis [CS524]: automatic, terminating, but may not be exact

  - Machine learning: automatic? terminating? exact?

- This lecture: verification via constraint solving

  - With Constrained Horn Clause (CHC)

  - Using SMT solvers

Horn Clause

A. Horn

- Clause: a disjunction of literals

  - E.g., $p \vee \neg q \vee \neg r$

- Horn clause: a clause with at most one positive literal

  - E.g., $\neg p \vee \neg q \vee r$ which is equivalent to $p \wedge q \rightarrow r$

- Horn clause logic: the basis of logic programming languages such as Prolog and Datalog

- Why Horn clause? Efficiency

  - Propositional Horn clause (HORNSAT): linear time

  - First-order Horn clause (e.g., Prolog): undecidable but efficient

- More details: CS402 (Introduction to Logic for Computer Science)

Constrained Horn Clause (CHC)

- A fragment of first-order logic

$$\forall x.\underline{\varphi} \wedge \underline{p_1(X_1) \wedge \cdots \wedge p_n(X_n) \rightarrow h(X)}$$

Constraint   **Predicates**

- φ: a constraint in a background theory T

  - E.g., $x + 1 = 2$ (Peano arithmetic)

Example
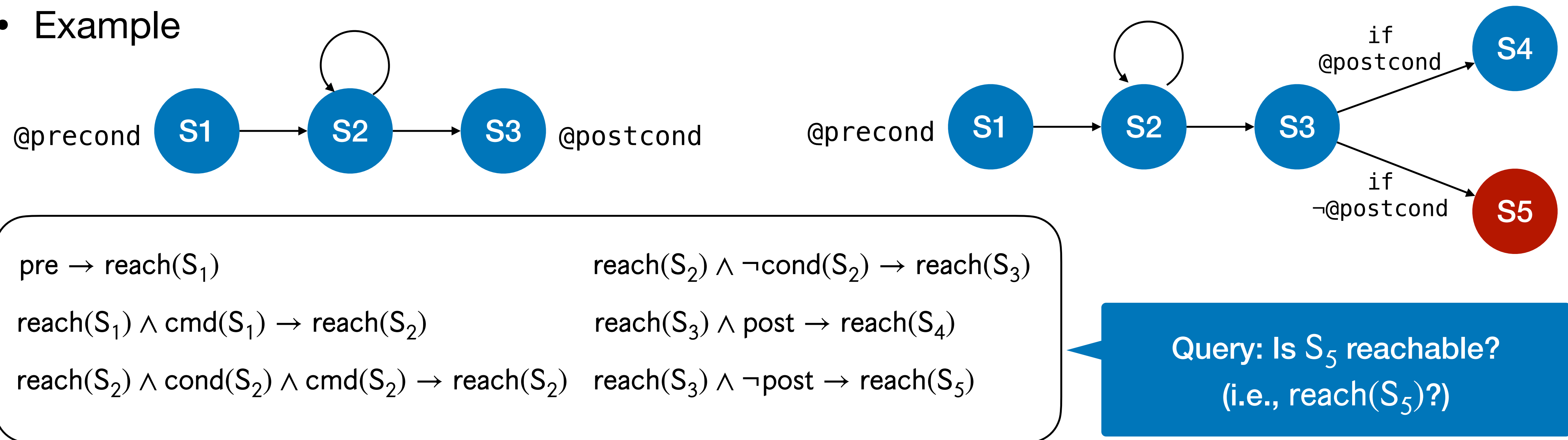
- Are the CHC formulas satisfiable? If so, what is P ?

$$P(0)$$
$$\forall x, x' . P(x) \wedge x < 10 \wedge x' = x + 1 \rightarrow P(x')$$

$$\forall x . x \leq 0 \rightarrow P(x)$$
$$\forall x, x' . P(x) \wedge x < 5 \wedge x' = x + 1 \rightarrow P(x')$$

Program Verification via CHC

- Given a program and a specification, generate verification conditions using CHC

- Check the satisfiability of the CHC formula using SMT solvers

- Idea: partial correctness check as a reachability problem

- Example

$\text{pre} \rightarrow \text{reach}(S_1)$

$\text{reach}(S_1) \land \text{cmd}(S_1) \rightarrow \text{reach}(S_2)$

$\text{reach}(S_2) \land \text{cond}(S_2) \land \text{cmd}(S_2) \rightarrow \text{reach}(S_2)$

$\text{reach}(S_2) \land \neg\text{cond}(S_2) \rightarrow \text{reach}(S_3)$

$\text{reach}(S_3) \land \text{post} \rightarrow \text{reach}(S_4)$

$\text{reach}(S_3) \land \neg\text{post} \rightarrow \text{reach}(S_5)$

Query: Is S_5 reachable?
(i.e., $\text{reach}(S_5)$?)

Language

- Program = control flow graph

- Node = basic block = list of commands (end with jump)

$$C \rightarrow \texttt{skip} \mid x := E \mid x := \texttt{input}() \mid \texttt{br } B \; l_1 \; l_2$$
$$\mid \texttt{goto } l \mid \texttt{assume}(E) \mid \texttt{assert}(E)$$
$$E \rightarrow n \mid x \mid E + E \mid E - E \mid E \times E \mid E \; / \; E$$
$$B \rightarrow \texttt{true} \mid \texttt{false} \mid E < E \mid E = E \mid \neg B$$

- Example:

```
Entry:
    x := input()
    assume(x > 1)
    assert(x == 0)
```

```
Entry:
    x := input()
    y := x - 1
    br x / 2 != 0 L1 L2
L1:
    assert(y != 0)
L2:
    skip
```
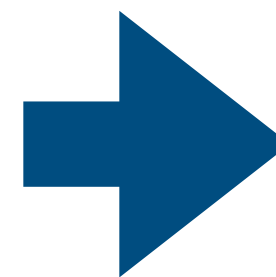
Specification

- Annotated in programs using assertions

- Checking an assertion = checking a reachability

  - Assertion is false = error state is reachable

- Example

```
Entry:
  x := input()
  y := x – 1
  assert(y != 0)
```

```
Entry:
  x := input()
  y := x – 1
  br y != 0 L1 L2
L1:
  skip
L2:
  assert false
```

State

- A predicate parameterized by values of variables defined so far

  - One relation per basic block

- Example

```
Entry:
  x := input()
  y := x - 1
  br y != 0 L1 L2
L1:
  skip
L2:
  assert false
```

Relations: Entry, L1(x, y), L2(x, y)

Reachable states:
Entry,
L1(2, 1), L1(3,2), L1(4, 3), …
L2(1, 0)

Verification Condition

- CHC formula: the relationship among all nodes + unreachability of the error node

- Loop invariants will be computed by the underlying solver (But not always! Why?)

- Example

```
Entry:
  x := input()
  y := x - 1
  assert(y != 0)
```

```
Entry:
  x := input()
  y := x - 1
  br y != 0 L1 L2
L1:
  skip
L2:
  assert false
```

The condition is SATISFIABLE
iff
there exists an erroneous input

$Entry$

$\forall x, y . Entry \wedge y = x - 1 \wedge y \neq 0 \rightarrow L_1(x, y)$

$\forall x, y . Entry \wedge y = x - 1 \wedge y = 0 \rightarrow L_2(x, y)$

$\exists x, y . L_2(x, y)$

Example

```
x := input();
assume(x < 10);
while(x < 10) {
  x++;
}
assert(x == 10);
```

⬇

```
Entry:
  x0 := input()
  assume(x < 10)
  goto Cond
Cond:
  x1 := φ [x0, Entry] [x2, Body]
  br (x1 < 10) Body End
Body:
  x2 := x1 + 1
  goto Cond
End:
  br (x1 = 10) Then Else
Then
  skip
Else:
  assert false
```

The condition is SATISFIABLE
iff
there exists an erroneous input

$Entry$

$\forall x \,.\, Entry \wedge x < 10 \to Cond(x)$

$\forall x \,.\, Cond(x) \wedge x < 10 \to Body(x)$

$\forall x \,.\, Cond(x) \wedge x >= 10 \to End(x)$

$\forall x, x' \,.\, Body(x) \wedge x' = x + 1 \to Cond(x')$

$\forall x \,.\, End(x) \wedge x = 10 \to Then(x)$

$\forall x \,.\, End(x) \wedge x \neq 10 \to Else(x)$

$\exists x \,.\, Else(x)$

Summary

- Automated program verification for partial correctness

  - Equivalently, checking reachability of error states

- Constrained Horn clause: a fragment of FOL

- Program verification using CHC

  - Verification condition = unreachability of error states

- Automatically solved by theorem provers